Universal Uniswap V3 Staking Contract

Qi Zhou, v0.1.2

June 2021

Abstract

In this paper, we present a liquidity mining reward computation for Uniswap V3 using cumulative functions. We prove that with the cumulative functions of liquidity and accumulated reward, the mining reward of any Uniswap V3 position can be computed in constant time. Further, we present an algorithm to show that given an update of a function over a contiguous discrete set, the cumulative function of the function can be updated by a binary tree implementation with at most $O(\log_2(N))$ operations.

1 Problem Statement

Definition 1 (Position)**.** *A position of a Uniswap V3 pool, p, is defined by a tuple*

$$(b_p, e_p, l_p), \tag{1}$$

where b_p and e_p are the begin and end price ticks of the position with $b_p < e_p \in \mathbb{Z}$, and $l_p \in \mathbb{R}$ is the liquidity of the position. Note that to simplify the setup, we assume $b_p, e_p \in \{0, \cdots, N-1\}$, where N is the cardinality of the price set.

Definition 2 (Active liquidity)**.** *Given the set of positions $\mathcal{P}$ of a Uniswap V3 pool, and the current price tick $q \in \{0, \cdots, N-2\}$, where the actual price lies in $[q, q+1]$, the active liqudity of the pool is*

$$al(\mathcal{P}, q) = \sum_{p \in \mathcal{P}} alp(p, q) = \sum_{p \in \mathcal{P}} l_p \times active(p, q), \tag{2}$$

where $alp(p, q) = l_p \times active(p, q)$ is the active liquidity of position p at price q with

$$active(p, q) = \begin{cases} 1 & \text{if } b_p \le q < e_p \\ 0 & o.w. \end{cases}. \tag{3}$$

Definition 3 (Reward of Liquidity Mining)**.** *Given the positions $\mathcal{P}_t$ of a Uniswap V3 pool, the prices q_t, and the liquidity mining rewards of the pool R_t over a*

time window $t \in [0, \cdots, T]$, the expected liquidity mining reward of a position at the end of the time window is

$$r(p) = \sum_{t=0}^{T} \frac{R_t \times alp(p, q_t)}{al(\mathcal{P}_t, q_t)}, \tag{4}$$

where we define $0/0 = 0$ in above equation, and we assume $p \in \mathcal{P}_t, \forall t$.

2 Efficient Reward Computation with Cumulative Function

Definition 4 (Cumulative Function)**.** *The cumulative function of $f(x), x \in \{0\} \cup \mathbb{Z}^+$ is defined as*

$$F(x) = \sum_{i=0}^{x} f(i). \tag{5}$$

Proposition 1. *Given the positions $\mathcal{P}$ of a Uniswap V3 pool, let us define*

$$f_{\mathcal{P}}^{(0)}(x) = \sum_{p \in \mathcal{P} | b_p = x} l_p, \tag{6}$$

and

$$f_{\mathcal{P}}^{(1)}(x) = \sum_{p \in \mathcal{P} | e_p = x} l_p, \tag{7}$$

then

$$al(\mathcal{P}, x) = F_{\mathcal{P}}^{(0)}(x) - F_{\mathcal{P}}^{(1)}(x). \tag{8}$$

Proof. Following Eqs. (5), (6), and (7), we have

$$F_{\mathcal{P}}^{(0)}(x) = \sum_{p \in \mathcal{P} | b_p \leq x} l_p, \tag{9}$$

and

$$F_{\mathcal{P}}^{(1)}(x) = \sum_{p \in \mathcal{P} | e_p \leq x} l_p. \tag{10}$$

Therefore, we have

$$\begin{aligned} F_{\mathcal{P}}^{(0)}(x) - F_{\mathcal{P}}^{(1)}(x) &= \sum_{p \in \mathcal{P} | b_p \leq x} l_p - \sum_{p \in \mathcal{P} | e_p \leq x} l_p && (11) \\ &= \sum_{p \in \mathcal{P} | b_p \leq x < e_p} l_p && (12) \\ &= \sum_{p \in \mathcal{P}} l_p \times active(p, x) && (13) \\ &= al(\mathcal{P}, x), && (14) \end{aligned}$$

which concludes the proof. □

Definition 5 (Accumulated Reward Function)**.** *Given the positions $\mathcal{P}_t$ of a Uniswap V3 pool, the prices q_t, and the liquidity mining rewards of the pool R_t over a time window $t \in [0, \cdots, T]$, the accumulated reward at price q at the end of the window is*

$$f^{(2)}_{\mathcal{P}_t}(q) = \sum_{t \in \{0,\cdots,T\} | q = q_t} \frac{R_t}{al(\mathcal{P}_t, q_t)}. \tag{15}$$

With the definition, we further have the following proposition.

Proposition 2. *The reward of a position during liquidity mining can be represented by the cumulative function of accumulated reward function as*

$$r(p) = l_p \times \left(F^{(2)}_{\mathcal{P}_t}(e_p - 1) - F^{(2)}_{\mathcal{P}_t}(b_p - 1) \right). \tag{16}$$

Proof. Plugging Eq. (15) into Eq. (5), we have

$$F^{(2)}_{\mathcal{P}_t}(q) = \sum_{t \in \{0,\cdots,T\} | q_t \leq q} \frac{R_t}{al(\mathcal{P}_t, q_t)}. \tag{17}$$

As a result, the right hand side of Eq. (16) is

$$\begin{aligned} l_p \times \left(F^{(2)}_{\mathcal{P}_t}(e_p - 1) - F^{(2)}_{\mathcal{P}_t}(b_p - 1) \right) &= l_p \times \sum_{t \in \{0,\cdots,T\} | b_p \leq q_t \leq e_p - 1} \frac{R_t}{al(\mathcal{P}_t, q_t)} && (18) \\ &= l_p \times \sum_{t \in \{0,\cdots,T\} | b_p \leq q_t < e_p} \frac{R_t}{al(\mathcal{P}_t, q_t)} && (19) \\ &= \sum_{t=0}^{T} \frac{R_t \times l_p \times active(p, q_t)}{al(\mathcal{P}_t, q_t)} && (20) \\ &= r(p) && (21) \end{aligned}$$

which concludes the proof. □

A couple of observations of interest are listed as follows.

- From Eq. (16), as long as we have computed the cumulative function of accumulated reward $F^{(2)}_{\mathcal{P}_t}$, then we could compute the reward of *any position* in constant time.
- To compute cumulative function of accumulated reward, as demonstrated in the next section, we could represent the cumulative function in a binary tree and update the tree when an accumulated reward point is changed with at most $O(\log_2(N))$ operations (worst case).
- When $\mathcal{P}_t$ is changed by **stake/unstaking** events, or q_t is changed by **priceUpdate** event over time (e.g., from time T to $T + 1$), accumulated reward must be updated accordingly. In addition, in the case of **stake/unstaking** events, the cumulative functions of liquidity (Eqs. (6) and (7)) also need to be updated with $O(\log_2(N))$ cost.

- Further, if $\mathcal{P}_t$ and q_t is unchanged over a time window, i.e., the denominator in the equation is fixed, since R_t is generally known over all the time, Eq. (15) can be batch updated over the time window.

To summarize, the proposed staking contract will maintain the following three cumulative functions:

$$F_{\mathcal{P}}^{(0)}(x), F_{\mathcal{P}}^{(1)}(x), F_{\mathcal{P}}^{(2)}(x), \tag{22}$$

to allow any staker quickly compute their rewards on chain, and when

- Staking/unstaking event happens, the contract will first update liquidity cumulative function $F_{\mathcal{P}}^{(0)}(x), F_{\mathcal{P}}^{(1)}(x)$ with the added/removed position p, and then update the cumulative function of accumulated reward $F_{\mathcal{P}}^{(2)}(x)$.
- PriceUpdate event happens, the contract will only update the cumulative function of accumulated reward $F_{\mathcal{P}}^{(2)}(x)$.

3 Efficient Cumulative Function Computation over Discrete Set

Given a update of $f(x), x \in \{0, \cdots, N-1\}$, we will demonstrate a binary tree implementation to fast compute its cumulative function $F(x) = \sum_{i=0}^{x} f(i)$ with at most $O(\log_2(N))$ operations (assuming $N = 2^D$ for simplicity).

Taking $D = 3$ as example, the tree will look like

```
        100          adding 100 with parent bits (0)
      /     \
    010     110      adding 010 with parent bits
    / \     / \
 001 011 101 111    adding 001 with parent bits
```

where a node of a tree contains two fields

- x: unique integer identifier of the node
- v: sum up of $f(x)$ in (last visited node x' from root such that $x' < x$, x]
  E.g.,
  - v of node $x = (010)_2$ represents the sum of $f(x)$ in $((000)_2, (010)_2]$
  - v of node $x = (101)_2$ represents the sum of $f(x)$ in $((100)_2, (101)_2]$
  - v of node $x = (111)_2$ represents the sum of $f(x)$ in $((110)_2, (111)_2]$

The tree supports two operations

- Add v to $f(x)$ such that $f(x) \leftarrow f(x) + v$. This operation will traverse the tree from root to the node with $node.x = x$ and add v to all the nodes in the path if $x \leq node.x$.

- Get $F(x)$ given x. This operation will traverse the tree from root to the node with $node.x = x$ and sum up all $node.v$'s if $x \geq node.x$ in the path.

A prototype written in python can be found here `https://gist.github.com/qizhou/b87d9b865102227dafe8281de447c43c`.